FUJITSU

FUJITSU Software
Agile+ Relief C/C++ V1.1.1

Manual for Notes

B1WD-3612-01ENZ0(00)
January 2023

Preface

Agile+ Relief C/C++(referred to below as Agile+ Relief) is an application that supports program review by analyzing source programs and header files written in C and C++.

This manual contains explanations of the messages that Agile+ Relief may output when it finds possible program errors. More information about these error messages may be found in the "Command Manual".

Contents

1. Notes for Source Analysis and Message Indications/Metrics

1.1 About Command Option

1. The filled text of option error will turn into --P text after -c-P option is specified for pgr5.

2. If analyzing result file (.ao) based on C++ source is specified as input file in pgr5, the normal execution cannot be guaranteed. Please do not use the analyzing result file based on C++ source as input file.

Sample:

```
$ pgr5 -c a.cc b.cc <-Check for a.cc and b.cc, then create a.cc.ao and b.cc.ao.
$ pgr5 a.cc.ao b.cc.ao <-Normal execution cannot be guaranteed.
```

3. For the files of group_change_file-Z option or those for check of message_select_file option within pgr5, the input of group name with half-width uppercase letters will not arise an error, it will be processed as half-width capitals. For example, while describing A in definition file for check, a group will not be indicated as an error.

4. When GB2312 is specified for --output_code option within pgr5, and Chinese is included in the message indication, it may be garbled.

5. Specify a group_change_file using -G option of pgr5 command, it may cause analysis cancel. If following error message is outputted, please open appropriate file and delete Message ID of applicable row.

PGRCDCHK_0410 Message ID @3 in line @2 of definition file @1 is not found.

@1: Filename @2: Line Number @3: Message ID

1.2 About Analysis

1. If @ found in variable name/function name, because it is not qualified with ANSI standard, the analysis cannot be performed.

Sample:

```
void func(void)
```

```
{
   int i@,funcA@(void);
}
```

2. Parse is not performable when the row of source program for parse exceeds 100K lines.

3. Parse will be terminated unexpectedly when hundreds of thousands of initial values are found in a source file or all imported include files.

Sample:

```
struct TAG a[ ]
{
0,1,2,3,4,~,1000000      /*The number of initial value is around 100,000.*/
};
```

Array a alike is frequently used.

4. When source file that takes header file and Type Library that Visual C++ 2005, Visual C++ 2008, Visual C++ 2010, Visual C++ 2012, Visual C++ 2013 or Visual C++ 2015 generated is analyzed, analyze is not performable. Please execute the analysis again after conversion the code system of header file and Type Library same as the sourcefile.

5. If a type under the same name as the nest type of the template parameter exits in template definition or declaration, it can not be analyzed.

Sample:

```
template < class T > class A { ~ };
template < class T > void F( A < typename T::A > ) { ~ }
```

6. If an operator ">" exits in template arguments, it can not be analyzed.

Sample:

```
#define N1 1
#define N2 2
template < int i > class T    { ~ } ;
T< ( (N1 > N2) ? N1 : N2 ) > t1 ;
```

7. It is not likely to be able to analyze it for the source program compiled specifying optional "-fpermissive" and "-trigraphs" for the compilation time by the source program compiled by gcc or g++ command.

1.3 About Message Indication

1.3.1 General Case for Message Indication

1. When loop counter is a floating decimal, the message cannot be output correctly because it is hard to determine whether loop has been finished for once.
Related message indications ID are pgr0000, pgr0060, pgr0520, pgr0522, pgr0524, pgr0539, pgr0541, pgr0667, pgr0679 and pgr0689.

Sample:

```
void func(void)
{
   int val = 0;
   int mode = 0;
   float f;
   int *ip = 0;
   for(f = 0.0;     /* Loop is surely finished, while the actual recognization is not so.*/
       f <6.0 ;
       f++)
      {
       ip = &val;
      }
      mode = *ip;      /* pgr0060 will be output */
}
```

2. When operator function is applied in actual calculation, because the process is performed as normal calculation other than by operator function, thus the message cannot be output correctly. Except the messages for declaration and suffix check, almost all message indications will be output for it.

Sample 1:

```
 class A {
```

```
        :
    int operator || ( int arg){ ~ }
        :
} x;
if ( x || i++ )     /*pgr0744 will be output for i++.*/
```

Sample 2:

```
class B {
        :
    int operator == ( int arg){ ~ }
        :
} x;
x == i;     /* pgr0038 will be output for ==. */
```

3. The message cannot be output correctly, because catch is not jumped in throw statement. The related message indications ID are pgr0000, pgr0054, pgr0060, pgr0074, pgr0520, pgr0522, pgr0524, pgr0541, pgr0667, pgr0679 and pgr0589.

Sample:

```
    if ( data == 1 ) {
        p = 0;
        throw( p );
    }
    *p = 1;     /* pgr0060 will be output.*/
```

4. Sometimes the incorrect conditions will not be indicated for floating decimal constant. The related message indications ID are pgr0030, pgr0037, pgr1222, pgr1223 and pgr1224..

Sample 1:

```
double d;
for(d = 5.0; d < 0.0; d += 1.0 )     /*pgr0030 will not be output.*/
```

Sample 2:

```
void func1(double x)
```

```
{
    if((x == 0.1 && x == 0.1)     /*pgr1222 will not be output.*/
```

5. Route check will not be indicated when && or || is found. The related message indications ID are pgr0000, pgr0007, pgr0039, pgr0050, pgr0051, pgr0054, pgr0055, pgr0060, pgr0074, pgr0339, pgr0520, pgr0522, pgr0524, pgr0528, pgr0539, pgr0541, pgr0667, pgr0679 ,pgr0689,pgr5011,pgr5021,pgr5031 and pgr5041.

```
Sample:
LPM mp = 0;
~
if ( func1( 10 )
     || ( xp->xa != (tbl *) &t1
     && xp->xa!= (tbl *) &t2
     && xp->xa != (tbl *) &t3
     && xp->xa != (tbl *) &t4))
   return func2(100);
~
switch(xp->xw)
{
   case 0:
   {
      PIMS y = xp->xp;
      hr = func3(y->ya, y->yb, &y->yc, y->yd, FALSE, (MESSAGE *) &mp);
   }
~
}
~
hr = mp->xa->GetNames(mp,      /* pgr0060 will not be output for mp. */
                      ptaga, lpid, lags, name, pname);
```

6. For the distinguishment from the parameter name of long type, incorrect message might be output when the parameter name is the same with tag name or type name. The related message indications ID are those for temporary parameters check.

```
class s{
public = s();
};
int func(long s[])      /* pgr0304, pgr0551 and pgr0553 will not be output. */
{
   return *s;      /* pgr0541 will not be output. */
}
```

7. When the result of &, |, ^, ?: operation whose size less than that of int type is assigned to the variable of the same type, for the check upon the operation result is performed according ANSI standard, the message will be output incorrectly. The related message indications for type comparison whose ID are pgr0351, pgr0369, pgr0372, pgr0373, pgr0383, pgr0397 and pgr0398.

```
Sample1:
char uc,uc1,uc2;
uc = uc1 | uc2;        /* pgr0351 will be output.*/


Sample2:
char func(int in)
{
   return (in > 0) ? 'a' : 'b';      /* pgr0369 will be output.*/
}
```

8. The message will be output incorrectly when universal character name is found in character const. The related message indications ID are pgr0016, pgr0019, pgr0020~pgr0022, pgr0066, pgr0067 and pgr0381.

```
Sample:
char x = '\u0090'; /*pgr0381 will be output.*/
if( x == '\u0090')    /*pgr0020 will be output.*/
```

9. Within cast expression, that is type expression, the type or expression is a pointer to function, and also when the pointer is qualified with const, no message will be output. The related message indication ID is pgr0727 and pgr0736.

```
Sample:
```

```
void func(void);
void (*const pf)(void) = func;
p = (void (*)(int))pf; /*pgr0727 will not be output.*/
```

10. No message will be output during the assignment to the members of structure or union. The related message indication ID are as follows : pgr0000, pgr0039, pgr0050, pgr0051, pgr0060(*), pgr0074(*), pgr0339, pgr0361, pgr0520(*), pgr0522, pgr0524, pgr0541, pgr0689(*), pgr0833.

* these message will not be output for C++ source files.

Sample:

```
void m(void)
{
    tag_t   t;
    t.next = malloc(100);
} /*pgr0524 will not be output for t.next.*/
```

11. For operator overload or member pointer of typedef declaration, the embedded string of indication cannot be output correctly. Indication can be operated normally, but embedded string will be incorrect. The related message indications ID are pgr0246, pgr0309, pgr0439, pgr0440 and pgr0638.

Sample:

```
class A
{
    :
};
typedef A::* AType;
AType F() ;
void F() {   /*pgr0440 The added character string is wrong */
        :
}
```

12. The message will not be output correctly, when the invocation of overload function is not match to its definition. The related message indications are all message indications that check the type.

Sample:

```
typedef int INT;

void f ( int i, char d )    {

}

int f ( INT i, INT i2 ) {

}

void func( )

{

      int i1,i2;

      f ( i1,i2 );      /*pgr0519 will not be output */

}
```

13. The message will not be output correctly, when the instantiation of template class is not match to its definition. The related message indications ID are pgr0103, pgr0104 and pgr2227.

Sample:

```
template<class T1,class T2 = char>class A;

template<class T1,class T2> class A {

   public:

      class IT { };

      typedef IT it;

};

template<class T> class B : public A<T> {

};

B<int>::it func ( B<int>::it bit ) {      /*pgr2227 will not be output */

}
```

14. The message will not be output correctly and wrong message will be output, when it is hard to resolve the variable in the situation of finding out the defect through tracing the path. The related message indications ID are pgr5011, pgr5021, pgr5031 and pgr5041.

Sample 1: the result of resolution for argument in the conditions

```
char *p1 = NULL;
char *p2 = NULL;
if ( p1 == NULL )
    return;
p2 = p1 ;    /* the result of above if statement (p1 is not NULL) is not considered. */
func( p2 ) ;   /* it is resolved that passing NULL to func.
                  If not to check NULL for argument in func, the message
                  pgr5011 will be output. */
```

Sample 2: calculation for the value of variable which is operated
(except =,==,!=,<,>,<=,>=).

```
char buff[10];
n=9;
buff[n+1]=0;   /* the message pgr5021 will not be output.
                  the value of index will not be calculated, if there exists operator+. */
m=9;
m++;
buff[m] = 0;    /* pgr5021 will not be output.
                  When calculation exists on the path, the value of subscript cannot
                  be evaluated */
```

Sample 3: calculation for the value of argument after function invocation

```
char buff[10];
func( &n );   /* the value of argument is updated by func. */
buff[n]=0;    /* the message pgr5021 will not be output.
                 the value of index will not be calculated, because n maybe
                 updated. */
```

Sample 4: calculation for the value of external variable

```
/* 4-1 */
int g ;
char buff[10];
void f1( ) {
    g = 10 ;
```

```
    buff[g]=0;     /* the message pgr5021 will be output */
    f2( );
}
void f 2( ) {
    buff[g]=0;     /* the message pgr5021 will not be output.
                      the value of index will not be calculated, because it is
                      unavailable to trace the value of external variable between
                      the functions. */
}

/* 4-2 */
char *p;
void get() {
    p = malloc(10);    /* the message pgr5031 will be output
                          The value between functions of external variable will not be
                          traced, so free cannot be detected */
}
void release() {
    free(p);
}
void main() {
    get();
    release();
}
```

Sample 5: calculation for the value of array element

```
void f ( ) {
    char buff[10];
    int   index[5];
    index[0]=10;
    buff[ index[0] ] =0;   /* the message pgr5021 will not be output.
                              the value of index will not be calculated, because it is
                              unavailable to trace every array elements. */
```

Sample 6: calculation for the value of structure member

```
/* 6-1 */
void f1(int *p) {
    int x = *p;          /* the message pgr5011 will not be output */
```

```
}
void f2() {
    struct str s;
    s.m = NULL;    /* The value of structure member will not be traced,
                      so it is unable to determine whether to pass NULL */
    f1(s.m);
}

/* 6-2 */
void f() {
     str.m = malloc(10);   /* the message pgr5031 will be output
                              The value of structure member will not be traced,
                              so free cannot be detected */
     free(str.m);
}
```

Sample 7: calculation for the value of return value of function and parameter

```
/*7-1*/
int* get_null() {
    int *p = NULL;
    return p;
}

void f1(int *p) {
    int x = *p;      /* the message pgr5011 will not be output */
}
void f2() {
    int *p = get_null();      /* The return value of function cannot be located */
    f1(p);
}

/* 7-2 */
buf[10];
int f1() {
    int i = 10;
    return i;
}
void f2() {
    int i;
    int x = f1();    /* It is unable to apply the return value of function to the range */
    if(i <= x) {
        buf[i];      /* the message pgr5021 will not be output */
```

```
    }
}

/* 7-3 */
void get(char **p) {
    *p = malloc(10);       /* the message pgr5031 will be output */
}
void f1() {
    char *p;
    get(&p);               /* The value of function parameter cannot be located */
    free(p);
}
```

Sample 8: ternary operator

```
void f1(int *p) {
    int x = *p;       /* the message pgr5011 will not be output */
}
void f2() {
    char *p;
    int x = 0;
    p = x ? f3() : NULL;       /* Ternary operator cannot be identified correctly. */
    f1(p);
}
```

Sample 9: judgment of path that is not executed due to condition statement

```
void f1(int *p) {
    int x = *p;       /* the message pgr5011 will be output */
}
void f2() {
    char *p = NULL;
    int x = 1;
    if(x == 3) {
        f1(p);     /* It is unable to identify it is the path that is not executed */
    }
}
```

Sample 10: calculation for the value of function pointer

```
typedef void(*PF)(int*);
void f1(int *p) {
    int x = *p;            /* the message pgr5011 will not be output */
}
void f2() {
    int* p1 = 0;
```

```
    PF pf1 = &f1;       /* The function pointer cannot be traced */
    pf1(p1);
}
```

Sample 11: assignment using comma operator and bracket operator

```
void f1(int *p) {
    int x = *p;       /* the message pgr5011 will not be output */
}
void f2() {
    int *p1;
    int *p2 = (p1, NULL);     /* It is unable to identify that NULL has been
                                 assigned */
    f1(p2);
}
```

Sample 12: The replicated pointer is returned

```
int func01() {
    int *buf;
    int *frc;

    buf = (int*)calloc(5, sizeof(int));    /* the message pgr5031 will not be output.
                                              The replicated pointer is traced by
                                              mistake, so free cannot be detected. */

    frc = buf;
    free(buf);
    return frc;
}
int main(void) {
    int rc;
    rc = func01();
    return 0;
}
```

15. Wrong message will be output if there is a member variable of reference type when finding the uninitialized member variables in constructor. The related message indications ID are pgr2266 and pgr2269.

Sample:

```
class A {
   public:
      A(int& x) : a(x), b(0) { }      /* The message pgr2266 will be output */
      int& a;
      int* b;
};
```

16. The expression and the type to use decltype, auto type specifier, lambda expression, and Rvalue Reference cannot be correctly detected. The related message indications are all message indications that check the type and the expression using the above-mentioned syntax.

Sample:

```
void func(int j) {
      char c;
      auto i = c;
      decltype(c) n;
      i = 256;             /* The message pgr0019 will not be output */
      n = 256;             /* The message pgr0019 will not be output */
      c = []{ return 256; }();      /* The message pgr0351 will not be output */
}
void rfunc(const int && r) {
      char c = (char)r;             /* The message pgr2261 will not be output */
}
```

17. The message will not be output correctly and wrong message will be output, when Agile+ Relief can not calculate the value that is cast from floating decimal to int type. The related message indications are all message indications that check the value of floating decimal that is cast to int type.

Sample:

```
int i, a[10];

for(i = 0 ; i < ((int)(8.1)) ; i++)

{ a[i] = 5;}            /* The message pgr0013 will be output */
```

1.3.2 Special Case for Message Indication

Message Indication ID	Message Indication	Contents
pgr0062	The correct value of the @2 operation in @1 might not be able to be obtained.	The message will not be output correctly, because compound literal ((Type){ ~ } is not expected in C99 grammar. Sample: long long LLint; void func(void) { if(LLint-x < (int){1}<<2) /* pgr0062 will not be output.*/ {} }
pgr0339	The value passed to the parameter @1 is not used.	If an operation of same constant, like 1*1, found in the conditional expression of the for statement, it may be indicated even the value of parameter is referenced after the for statement. Sample: void func1(int para) { int num; for(num = 0; num < (1 * 1); num++){ } if(para != 0) /*Indicated*/
pgr0436	The variable definition @1 and the variable definition in line @3 of @2 are of different types (@4, @5).	It will be indicated when there are two casts, and the cast type is used to declare a member of the same name and different type. Sample: ap = (char *)&(((struct{char m1; unsigned int m2;}*) 0)->m2); ap = (char *)&(((struct{char m1; unsigned char* m2;}*) 0)->m2); /*m2 is indicated. */
pgr0506	The array @1 is initialized with the string @2 which exceeds its size.	When[wchar_tiBuf[sizeof(L"123456")/sizeof(wchar_t)] = L"123456";] is used as the initialization of wide char const, pgr0506 will be output. Sample: typedef struct TEST_STRUCTURE_ {

		char id[6]; char * title; char * button[14]; } TEST_STRUCTURE; TEST_STRUCTURE scr[2] = { { "0000", "TestTest", { "TestTest",/* 0 */ "TestTest", "TestTest", "TestTest", "TestTest", "TestTest",/* 5 */ "TestTest", "TestTest", "TestTest", "TestTest", NULL, /* 10 */ } }, { "1000", "TestTest", /*X_pgr0506*/ When the initialization of wide char const is used as [wchar_t iBuf[sizeof(L"123456")/sizeof(wchar_t)] = L"123456";], pgr0506 will be output. Sample: wchar_t iBuf[sizeof(L"12345")/sizeof(wchar_t)] = L"12345"; /*X_pgr0506*/

pgr0522	The variable @2, the resources of which are reclaimed in line @1, might be referenced.	It will not be indicated when two variables pointing to the same space like y=x; free(y); free(x);, and double release are done. Sample: char *x, *y; x = (char *)malloc(len); y = x; free(y); free(x); /* pgr0522 will not be output.*/
pgr0524	Resources allocated with the function @2 in line @1 may not have been deallocated.	It will be indicated when the return value of resource reclaiming function is assigned to the space pointed by the parameter of reference type or pointer type. Sample: #include <stdlib.h> typedef char * STR; void func(STR & x) { x = malloc(100); return; /* Indicated*/ } It may be not indicated when more than two function calls, like malloc or fopen, are found in a function definition. Sample: pData = malloc(size); if(pData == NULL) { return 1; } fp = fopen(fileName, "wb"); if(fp == NULL) { return 1; /* pData will not be indicated. */ } It will be indicated when a pointer (p1) of space obtained by realloc function is assigned to another pointer (p2), and the space pointed by p2 is released. p1 = malloc(10); if((p2 = realloc(p1,100)) == NULL){ return -1; }

		p1 = p2; free(p1); return 0; /* Indicated */
pgr0679	The union member @2 whose value is assigned in line @1 is referenced by another member @3 of the union.	It will not be indicated for unnamed union variable. Sample: static union { int g_ui1; signed char g_uc1; }; g_ui1 = 0; c = g_uc1; /* Not Indicated*/
pgr0689	The variable @2 compared with the 0 address in line @1 might access the 0 address.	It will be indicated when a variable with a value other than 0 assigned is compared with zero. Sample: char *buf = new char[2]; char *p=0; if(x) { p = buf; /*A value other than 0 is assigned to p.*/ *p=0; /* Indicated*/ } if(p == 0) /* Compared with 0.*/ It will be indicated when referenced other than being zero is recorded in loop. Sample: for(i = 0; i < 10; i++) { if(p != 0) { p++; /*indicated*/ It will be indicated when the pointer variable and zero is compared in the expression to the right of && or ||, while the pointer variable and a value is compared in the left one. Sample: if(p != (char*)-1 && p != 0) /* Indicated */

pgr0692	An infinite loop might result because type width of the loop counter @1 (of type @2) is less than the type width of the expression @3 (of type @4) with which it is compared.	It will be indicated when the type width of loop counter is less than that of int type, the result of comparison expression with loop counter will be judged to be of int type. Sample: short x; short i; for(i = 0; i < x - 1; i++) /*Indicated */
pgr0815	The name of @1 and @4 at the line @3 of file @2 are only different in case.	If the identifier declared in standard include file and the identifier with different uppercase/lowercase are defined, the message might not be correctly output. Sample: Filename : stdio.h void printf( ); Filename : sample.c void PrintF( ); /* Not Indicated */
pgr0818	An element of object @1 is initialized more than once.	If the structure and the array, etc. are initialized by the nest and the same element is initialized two or more times, the message might not be correctly output. Sample: int ary1[3][3] = { {[2]=1, [2]=2} }; /* Not Indicated */
pgr0833	The return value "@3" of function "@2" at line @1 point to a static field, it is an error to free it by @4 "@5".	If a certain function is called, it will not be indicated when this function returns to the pointer that points to static scope as an initializer. Sample: struct tm *localtimer = localtime(&timer); free(localtimer); /* Not Indicated */
pgr0869	The index @2 of array @1 may exceed the range of array.(array declaration : line @4 of @3)	It will not be indicated when the upper limit of subscript range cannot be judged. Sample: char array[10] ; int i; if(i > 5) { array[i]; /*Not Indicated*/ }

pgr5011	The argument @3 at the line @2 of @1 accessed to 0 address. (Route: @4)	It will not be indicated when referring the return value that maybe access the 0 address. Sample: void x (int * p) { *p = 1 ; } int* y ( ) { return NULL; } void z ( ) { int *p = y( ) ; x (p); /* Not Indicated */
pgr5021	The argument @3 at the line @2 of @1 accessed outside range of array. (Route:@4)	When passing the subscript in the parameter of a function, the range check of subscript is ignored. Sample: void x (int vno) { int TB[5]; if (vno < 5) { TB[vno]; /* Indicated */ } /* Subscript rage MIN=10 */ } /* MAX=10 */ void y ( ) { int number = 10; x (number); } When the subscript referenced by array is not variable, the range check of subscripts ignored. Sample: char buf[10]; int i = 9; buf[i+1]; /* Not Indicated */ When referencing array through pointer, it will not be indicated since the range of array cannot be obtained. Sample: char buf[10]; int *x = buf; int i = 10;

		x[i]; /* Not Indicated */ In case of dynamic array, it will not be indicated since the format of non-malloc (sizeof (type)* constant) cannot obtain the size of array. Sample: char *buf = malloc(10); /* since it is not in malloc(sizeof(char) * 10) the array size cannot be obtained. */ int i = 10; buf[i]; /* Not Indicated */ Since the size of dynamic array does not consider the size of type, it may indicate incorrectly. Sample: char *buf = malloc(sizeof(int) * 10); /* The array size is judged as 10. */ int i = 10; buf[i]; /* Indicated */

pgr5041	The resource allocated by @1 has been freed repeatedly. (Route: @2)	When repeated free occurred in loops, it will not be indicated. Sample: char *p = malloc(10); /* Not Indicated */ for(i=0; i<2; i++) { free(p); /* It is unable to judge it has been freed twice. */ } When free appears in the switch statement, it may not indicate sometimes. Sample: char *p = malloc(10); /* Not Indicated */ switch(i) { case 1: free(p); default: free(p); } When the object to be copied and the pointer of object after copying are freed separately, it will not be indicated. Sample: char *p1 = malloc(10); /* Not Indicated */ char *p2 = p1; free(p1); free(p2);

1. 4 About Metrics

Notes for metrics output by pgrmetrics command are as follows:

1. The number of nest is not counted in try and catch grammar (also inside the grammar).

Sample:

The number of nest in following source is 0.

void func(int a)

```
{
    try{
        if( a==1)
        { throw 1; }
    }
    catch(int x)
    { }
}
```

2. Notes for GUI Functionality

1. The insufficient disk space might lead to no response or the project file damage during parsing. Please be sure the space is enough before execution.

When the project file is damaged, it will be saved in the state of intact.

project directory \project filename.pg2.bak.

By copying the file to the following file, sometimes, the project file can be restored.

project directory \project filename.pg2

2. Select [System Option Settings] from [Settings], if no change is required, you can also click [Apply] button to leave (GUI-0156).

3. Select [System Default Option Settings] from [Settings], or [File Default Option Settings] from [Macro Settings], and press Alt key and L key at the same time, then press arrow key, sometimes the choice of macro name already registered is not available. It is the same with [Name & Token Settings].

4. In Windows, input the directory name following the drive name, while in Linux, input following /. If no input is done, the normal operation cannot be guaranteed.

5. In the dialogbox of file or directory selection, it may be unstable when network computer, my computer, my document or desktop is chosen. In addition, for network assets, please do drive distribution before use.

6. In [System Option Settings] of [Settings] menu, or [Extension Settings] of [System Default Option Settings], if all extensions are deleted from [Extension List], the parse is not performable.

7. When [Apply Pre-compiler Header] is selected in [Pre-compile Header Settings] of [Project Option Settings] from [Settings] menu, the following problems may arise:

- Choose any of [C] column or [C++] column, even Alt key and A key are pressed at the same time, select all is not performable.

- Click [Select All] button and then click [Apply] or [Yes], if C codes and C++ codes coexist within the project, select all is not performable.

Besides, when [Apply Pre-compile Header] is not chosen, [Select All] can be selected, then a tick can be done on all items.

8. Choose [Pre-compile Header Settings] of [System Default Option Settings] from [Settings] menu, even a tick is done on [For "stdafx.h"] in [C] column, it will not be a pre-compile object.

9. In the link of 'Help', when clicking "Return" button, the previous page cannot be accessed occasionally.